QTL DETECTION FROM STOCHASTIC PROCESS BY BAYESIAN HIERARCHICAL REGRESSION MODEL

Yi Chen

A Thesis Submitted to the
University of North Carolina Wilmington in Partial Fulfillment
of the Requirements for the Degree of
Master of Science

Department of Mathematics and Statistics

University of North Carolina Wilmington

December 12, 2007

Approved by

Advisory Committee

_____ _____ _____

_____
Chair

Accepted by

_____
Dean, Graduate School

ABSTRACT

The problem of identifying the genetic loci contributing to variation in a quantitative trait (called QTL) has been researched for a number of years, and is a growing field in statistical genetics[10]. Most research focuses on the problem with only one observation per genotype. For years, plant biologists have condensed replicates within lines to one genotype to use these conventional methods. In this paper we extend and apply one of the most widely used Markov Chain Monte Carlo Model Comparison(MC^3) algorithms, incorporated in a Bayesian hierarchical regression setting. This algorithm is then applied to simulation data in order to validate the model. Use of *Posterior Model Probability* and *Activation Probability* will be used for model comparison. Furthermore, based on *Acceptance Probability*, we perform stochastic search through the model space to identify potential QTL.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Susan Simmons, for her great insight, guidance, patience, encouragement and being there for me every step of the way. This could not have been possible without her. I appreciate Dr. Karl Ricanek for helping me improve the algorithm designing and coding. I would also like to thank Dr. Edwards Boone and Dr. Ann Stapleton, for assisting me in my understanding of the definitions and theories relevant to this thesis.

I also really appreciate all the faculty members and fellow students in the Department of Mathematics and Statistics and other departments for sharing their love and knowledge of mathematics and statistics with me. Last but not least, I would like to give the credits to my parents and all friends.

1 INTRODUCTION

1.1 Overview

A quantitative trait locus, also known as a QTL, is a region of DNA that is associated with a particular phenotypic trait. By definition, a quantitative trait is either continuous, such as yield of crop, or discrete, for example, cotyledon opening. The QTL on a genome is a location contributing significantly to the variation of a quantitative trait.

Though not necessarily genes themselves, QTL are stretches of DNA that are closely linked to the quantitative trait. Furthermore, a single phenotypic trait is usually determined by many QTL, which are often found on different chromosomes[15, 29]. And most traits of interest are governed by more than one gene. Therefore, defining and studying the entire locus of genes related to a trait gives hope of understanding what effect the genotype of an individual might have in the real world. Knowing the number and effect of such loci help us to understand the genetic basis of traits, and of their evolution in populations over time[1, 4, 6, 18]. Moreover, it may also tell which traits are controlled by a few genes of large effect as opposed to many genes of small effect, so that knowledge of these loci may aid in the design of selection experiments to improve these traits.

In fact, there are already some successful examples showing us benefits of identifying associations between regions on the genome and a trait. In 1989 a breakthrough was made for the disease of cystic fibrosis, although this is not quantitative trait, the benefits of knowing this location are profound. Location (or locus) 7q31.2(Figure 1) is the location of the single gene responsible for the disease. And it is located in region q31.2 on the long (q) arm of human chromosome 7[27]. The disease arises

when an individual has two recessive copies at this location, while an individual with one dominant and one recessive is said to be a carrier of the disease. Hence, we could make use of genetic screening to determine such disease. A good example of associating regions on the genome with a quantitative trait is the Green Revolution in which wheat and rice were genetically mutated. Through mutating genes, such as *sd1*, researchers are able to produce crops resistant to lodging and produce significant increases in yields[28]. Even though we already know lots of benefits concerning QTL, the problem of detecting QTL is still under research, and the methods available are increasing. In the past three decades we have witnessed an increased interest in QTL detection. The simplest approach, with data on an experimental cross, is to perform an analysis of variance (ANOVA) on each marker[1]. Unfortunately, when this approach was first introduced, there were not many markers on the genetic map, and ANOVA approaches were shown to be modestly effective[1, 4]. In addition, this method usually looks at one marker at a time, and disregards information from other markers. More power can be gained by considering more than one marker at a time, but the question becomes which markers should be modeled together?

For overcoming ANOVA weaknesses, one method used was interval mapping, which uses mapping distances and creates pseudo-markers between existing markers[1]. Many methods have been proposed throughout the 1990's that use interval mapping, such as the most popular method Composite interval mapping(CIM) which is developed by Jensen[16, 17] and Zeng[18, 19]. By using interval mapping and multiple regression on marker genotypes, CIM is very useful in identifying QTL in one-dimensional search. A number of software packages have been published and used, such as the rQTL package developed by K.W. Broman, John Hopkins[4], and QTL Cartographer by Z.-B. Zeng's group, North Carolina Sate University[2, 16, 17],

both of which are practical tools for QTL detection. Some researchers view identifying QTL as a model selection problem, and use strategies from multiple regression analysis, such as backward elimination based on AIC and stepwise selection[1]. A number of other methods are available, including Bayesian analysis methods, or use of genetic algorithms[1, 20]. Regardless of whether or not interval mapping is used, most methods use a regression setting when trying to determine which markers associate with the genetic trait. And this idea invokes the assumption of additivity.

This research provides a method to detect QTL in plant experiments. Plant QTL experiments take less time and cost less money than human and mice experiments, while the benefits are just as significant and useful. For example, we could improve the immunity of plants against different diseases, and thus increase yield to feed the increasing population. Plus, it is easier to control other factors which may affect the experiment, including the design, environment, reproduction, and etc. However, plant experiments have more complexity since replicate observations by independent genetic clones could be present for each genotype.

As genetic maps become more dense in plant experiments, a shift away from interval mapping has been observed[6]. The more dense the genetic maps become, the less effective interval mapping is. In fact, some of the recent genetic maps have too many markers for the conventional interval mapping systems. In addition, there are still some other serious issues which should be considered for QTL detection in plant experiments, such as the hierarchical structure of the data which increases computational complexity. Because most of the current methods can handle only one observation per genotype, most plant researchers average the observations within each line. The averaging leads to loss of information regarding the variance within each line[12]. Considering these issues, we model the data by a Bayesian hierarchi-

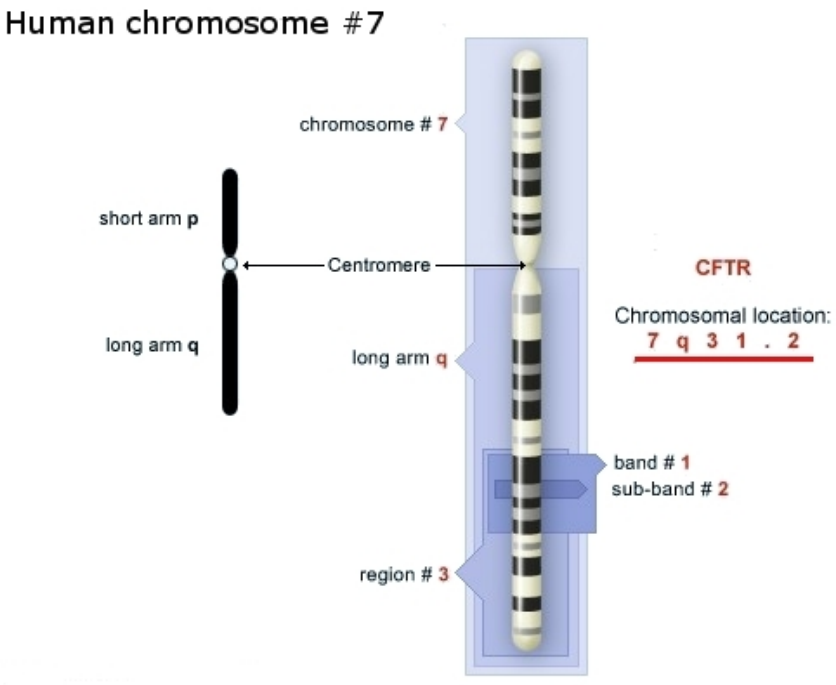

Figure 1: Human chromosome 7

cal model, and use a stochastic MC^3 algorithm to search through possible regression models in order to locate markers which are potential QTL.

1.2 Model Selection and Search Algorithm

There has been an increase in the consideration of model search algorithms for QTL detection lately[10]. Model selection can be viewed as the principal problem in identifying multiple QTL methods[20]. Considering the case of dense markers, relatively complete genotype data, and assumption of additivity, the question becomes which of the markers should be included in the model[20].

Considering the goal, our algorithm focuses on how to efficiently search through different models in the complex model space and how to select better linear models by model comparison in order to locate the best subset of markers as candidate QTL. Unlike usual model selection methods, such as forward, backward, or stepwise regression technique, our algorithm makes use of stochastic process to search different possible models in the model space. More importantly, using Bayesian modeling, the new algorithm is able to handle multiple replicates within a genotype situation and to locate all potential QTL simultaneously, no matter how many markers associate with the quantitative trait (as long as the number of markers in the model is less than the number of lines). Two criterion are very important to the new algorithm

- Criterion for comparing and selecting better models
- Strategies for efficiently searching through the large model space.

To define these two critical criterion of our algorithm, we will introduce **Posterior Model Probability** $P(k_q|D)$, **Model Selection Vector** $\vec{M}$, **Activation Probability** $p(\beta_j \neq 0|D)$, and **Acceptance Probability** α_{ji}. *Section 3* describes these quantities and how they are used to answer the critical criterion stated above. Before introducing the quantities, we will introduce the background and assumptions for our research in *Section 2*. Then, we will apply our algorithm to a simulated data set, and display the output from the analysis in *Section 4*. Finally, we will discuss our algorithm advantages and identify some issues for future improvement in *Section 5*.

2 BACKGROUND

2.1 Bayesian Statistics and Hierarchical Modeling

One of the fundamental differences between Bayesian statisticians and frequentists is that Bayesian statisticians regard parameters as random quantities. Unlike frequentists who assume that parameters are fixed, unknown quantities, Bayesian statisticians assume parameters are random variables with a probability distribution. Bayesians choose a prior distribution for the parameters which should be a first guess as to what the probability distribution of the parameter in question is. Prior distributions can be vague or noninformative when not much prior information is known, or more explicit if experts are able to provide information on the parameters. Information from the sample is combined with the prior distribution, and by using Bayes' Theorem, an updated or posterior distribution for the parameter is obtained. Assuming the unknown parameter is θ, $p(\theta)$ is called the prior probability distribution, which is interpreted as the knowledge about the parameter before combining it with the information details from the samples[24]. The distribution of the sample data is represented as $p(Y = y|\theta)$. Using the prior probability distribution and the information from the data, the posterior probability can be found by Bayes' theorem

$$p(\theta|Y = y) = \frac{p(Y = y|\theta)p(\theta)}{\int\limits_{\Theta} p(\theta)p(Y = y|\theta)d\theta} \tag{1}$$

where Θ is the sample space of the the parameter θ. This posterior probability distribution summarizes the knowledge about the parameters after combining the prior information with the sample information. A Bayesian analysis is based on inferences from the posterior distribution[5, 24, 25]. The Bayesian approach is distinct with respect to its flexibly in both incorporating prior information and using posterior probabilities[26].

Bayesian hierarchical modeling is one of the best approaches in dealing with the situation when the observed data involves multiple levels, such as QTL detection in plant experiment. For example, suppose the observed data y_{ij} is from a plant experiment, where y_{ij} is the j^{th} crop yield from the i^{th} genotype. In the first level, the distribution of observed data from any i^{th} genotype is assumed to be independently distributed with a mean represented by the parameter θ_i and a variance σ_i^2 which are both believed to be connected to the quantitative trait. The probability density function of the data given the parameters θ_i and σ_i^2 is written as $p(y_{ij}|\theta_i, \sigma_i^2)$, and is known as the likelihood function. By a linear regression model, the mean θ_i is assumed to be a normal distribution with some other parameters in the second level. Therefore, the Bayesian hierarchical model constructs a relationship between multiparameters by way of the layered data structure[26].

2.1.1 QTL Bayesian Hierarchical Regression Model

For our QTL algorithm, the Bayesian hierarchical model is built in the following fashion(Figure 2):

Level 1. For any $i^{th}(i = 1 \dots L)$ genotype, where L is the number of total genotypes in the experiment, there are n_i observations. n_i's must be greater than 2 and they are not necessarily the same for different genotypes. A natural choice for the observed data y_{ij}, where $j = 1 \dots n_i$, is the normal distribution conditional on the mean θ_i and variance σ_i^2:

$$y_{ij}|\theta_i, \sigma_i^2 \sim N(\theta_i, \sigma_i^2) \text{ for } i = 1 \dots L; j = 1 \dots n_i.$$

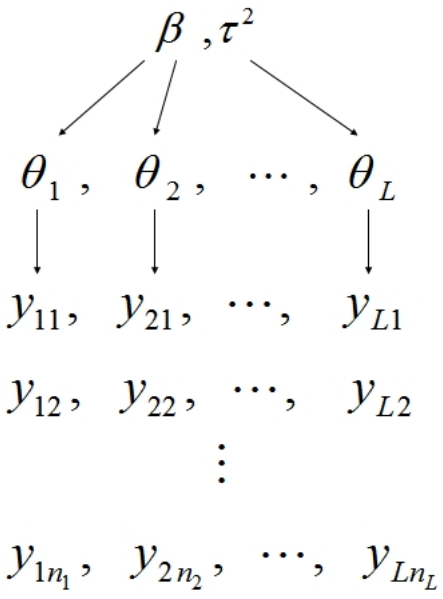

Figure 2: Bayesian Hierarchical Model

Level 2. With the assumption of additivity, a linear regression model includes variables which represent different candidate markers[21]. One possible linear regression model in plant experiment with L genotypes, we define

$$\theta_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \ldots + \beta_M x_{Mi} + \epsilon_i$$

Where M is the number of candidate markers included in the model, $i = 1, \ldots, L$, and $x_{ki}(k = 1, \ldots, M)$ are origin information of the markers which are contained in a genotype matrix, X. Therefore, for any i^{th} mean, $\theta_i(i \in [1, L])$, it is assumed to be normally distributed given the mean based on the genotype matrix X, $\vec{\beta}$ and variance τ^2:

$$\theta_i | \vec{\beta}, \tau^2 \sim N(X\vec{\beta}, \tau^2)$$

In addition, we set the prior distribution for parameters, $\vec{\beta}$, σ_i^2, and τ^2[27].

$$\begin{aligned} p(\beta_j) &\sim N(0, 100) \\ p(\sigma_i^2) &\sim Inv - \chi^2(1) \\ p(\tau^2) &\sim Inv - \chi^2(1) \end{aligned}$$

For both σ_i^2 and τ^2, the choice for the degrees of freedom of the $Inv - \chi^2$ is **1**, since this creates a prior distribution with infinite variance[10].

Considering the hierarchical model and prior distribution assumptions, there are 4 full conditional posterior distributions, $p(\vec{\theta}|\vec{\beta}, \vec{\sigma^2}, \tau^2, y)$, $p(\vec{\sigma^2}|\tau^2, \vec{\theta}, \vec{\beta}, y)$, $p(\vec{\beta}|\vec{\theta}, \vec{\sigma^2}, \tau^2, y)$, $p(\tau^2|\vec{\theta}, \vec{\beta}, \vec{\sigma^2}, y)$[12]. These 4 full conditional posterior distributions can be derived

by Bayes' theorem as following[12],

$$\begin{aligned} p(\vec{\theta}|\vec{\beta},\vec{\sigma^2},\tau^2,y) &= \frac{p(\tau^2,\vec{\theta},\vec{\beta},\vec{\sigma^2}|y)}{p(\tau^2,\vec{\beta},\vec{\sigma^2}|y)} \\ &\propto exp\left\{\sum_{i=1}^{L}\frac{-1}{2\left(\frac{1}{\frac{1}{\tau^2}+\frac{n_i}{\sigma_i^2}}\right)}\left(\theta_i-\frac{\frac{\vec{x_{i.}}\vec{\beta}}{\tau^2}+\frac{\sum_{j=1}^{n_i}y_{ij}}{\sigma_i^2}}{\frac{1}{\tau^2}+\frac{n_i}{\sigma_i^2}}\right)^2\right\} \end{aligned} \tag{2}$$

$$\theta_i|\tau^2,\vec{\beta},\vec{\sigma^2},y \sim N\left(\frac{\frac{\vec{x_{i.}}\vec{\beta}}{\tau^2}+\frac{\sum_{j=1}^{n_i}y_{ij}}{\sigma_i^2}}{\frac{1}{\tau^2}+\frac{n_i}{\sigma_i^2}},\frac{1}{\frac{1}{\tau^2}+\frac{n_i}{\sigma_i^2}}\right)$$

$$\begin{aligned} p(\vec{\sigma^2}|\tau^2,\vec{\theta},\vec{\beta},y) &= \frac{p(y|\vec{\theta},\vec{\sigma^2})p(\vec{\sigma^2})}{\int p(y|\vec{\theta},\vec{\sigma^2})p(\vec{\sigma^2})d\vec{\sigma^2}} \\ &\propto \prod^{L}(\sigma_i^2)^{-(\frac{\sigma_0^2+n_i}{2}+1)}exp\left\{-\sum_{i=1}^{L}\left(\frac{1}{2\sigma_i^2}\right)\left[\sum_{j=1}^{n_i}(y_{ij}-\theta_i)^2+1\right]\right\} \end{aligned} \tag{3}$$

$$\sigma_i^2|\tau^2,\vec{\theta},\vec{\beta},y \sim Inv-Gamma\left[\frac{\sigma_0^2+n_i}{2},\frac{\sum_{j=1}^{n_i}(y_{ij}-\theta_i)^2+1}{2}\right]$$

$$\begin{aligned} p(\vec{\beta}|\vec{\theta},\vec{\sigma^2},\tau^2,y) &= \frac{p(\vec{\theta}|X\vec{\beta},\tau^2)p(\vec{\beta})}{\int p(\vec{\theta}|X\vec{\beta},\tau^2)p(\vec{\beta})d\vec{\beta}} \\ &\propto exp\left\{-\frac{1}{2}\left[\vec{\beta}-\left(\frac{I}{100}+\frac{X'X}{\tau^2}\right)\frac{X'\vec{\theta}}{\tau^2}\right]'\left(\frac{I}{100}\frac{X'X}{\tau^2}\right)\right. \\ &\quad \left.\left[\vec{\beta}-\left(\frac{I}{100}+\frac{X'X}{\tau^2}\right)\frac{X'\vec{\theta}}{\tau^2}\right]\right\} \end{aligned} \tag{4}$$

Where $\sigma_0^2=1$, and I is an identity matrix.

$$\vec{\beta}|\vec{\theta},\vec{\sigma^2},\tau^2,y \sim N\left[\left(\frac{I}{100}+\frac{X'X}{\tau^2}\right)\frac{X'\vec{\theta}}{\tau^2},\left(\frac{I}{100}+\frac{X'X}{\tau^2}\right)^{-1}\right]$$

$$\begin{aligned} p(\tau^2|\vec{\theta},\vec{\beta},\vec{\sigma^2},y) &= \frac{p(\tau^2,\vec{\theta},\vec{\beta},\vec{\sigma^2}|y)}{p(\vec{\theta},\vec{\beta},\vec{\sigma^2}|y)} \\ &\propto (\tau^2)^{-(\frac{L+\tau_0^2}{2}+1)} exp\left\{-\frac{[(\vec{\theta}-X\vec{\beta})'(\vec{\theta}-X\vec{\beta})+1]/2}{\tau^2}\right\} \end{aligned} \tag{5}$$

where $\tau_0^2 = 1$, and L is the number of genotypes in the plant experiment

$$\tau^2|\vec{\theta},\vec{\beta},\vec{\sigma^2},y \sim Inv-Gamma\left[\frac{L+\tau_0^2}{2},\frac{(\vec{\theta}-X\vec{\beta})'(\vec{\theta}-X\vec{\beta})+1}{2}\right]$$

2.2 Stochastic Process by MCMC and Gibbs Sampler

The implementation of Bayesian analysis has received much attention since the 1990's when computer and numerical algorithms could compute posterior probabilities [11, 26]. Monte Carlo Markov Chain(**MCMC**) methods are a class of algorithms for generating samples, especially in high-dimensional space. Each sample is generated by using information from the previous sample to produce a Markov chain to approach to the target distribution, known as Stationary distribution. Hence, only the current state is necessary for generating a subsequent state or states in such a process. Furthermore, the Markov chain generated by MCMC algorithm describes at successive times the states of the system[1].

One particular MCMC method, **Gibbs Sampler**, is a powerful numerical instrument that is widely used for this broad class of Bayesian analysis, especially when

multiple variables are involved[9, 11]. The algorithm uses parameter-by-parameter updating. For example, in our problem, there are 4 parameter vectors involved in the posterior distribution: $\vec{\beta}$, τ^2, $\vec{\theta}$, $\vec{\sigma^2}$. Using the Gibbs Sampler, samples from the posterior can be generated by

$$
\begin{array}{lrcl}
1. & \beta_j^{(t+1)} & \sim & \mathrm{p}(\beta_j|y, \tau^{2(t)}, \theta_i^{(t)}, \sigma_i^{2(t)}) \\
2. & \tau^{2(t+1)} & \sim & \mathrm{p}(\tau^2|y, \beta_j^{(t+1)}, \theta_i^{(t)}, \sigma_i^{2(t)}) \\
3. & \theta_i^{(t+1)} & \sim & \mathrm{p}(\theta_i|y, \beta_j^{(t+1)}, \tau^{2(t+1)}, \sigma_i^{2(t)}) \\
4. & \sigma_i^{2(t+1)} & \sim & \mathrm{p}(\sigma_i^2|y, \beta_j^{(t+1)}, \tau^{2(t+1)}, \theta_i^{(t+1)})
\end{array}
$$

This auto correlated sequence, called Gibbs sequence, eventually "forgets" the initial stage of the chain, such as $\vec{\beta}^{(0)}$, $\tau^{2(0)}$, $\vec{\theta}^{(0)}$,$\vec{\sigma^2}^{(0)}$, and converges to a stationary posterior distribution $p(\vec{\beta}, \tau^2, \vec{\theta}, \vec{\sigma^2}|y)$. And this stationary distribution is the target distribution we are trying to simulate[11, 12]. However, a key issue in the successful implementation of the sampler is the number of run(steps) until the chain approaches stationarity, which is known as the burn-in period. Therefore, the first few thousands samples should be thrown out[11]. Since we have drawn a large number of samples, the full posterior probabilities, which are computationally difficult to calculate because of high-dimensional functions, can be approximated by averaging samples, which is referred to as the Monte Carlo Integration[11].

In addition to fitting hierarchical models to complex data sets, MCMC algorithms can also be applied to stochastically search for the best model. In QTL detection problem, the number of possible additive models is 2^M if there are M genetic candidate markers. For example, there are 38 candidate markers in Bay-0 $\times$ Shahdara recombinant inbred lines from Arabidopisis thaliana, giving 2^{38}, or 274,877,906,944, possible regression models. A stochastic search is one of the best ways to deal with a very, very large model space. During the stochastic search, an acceptance proba-

bility α is used to decide how to move through the parameter space, and is defined as:

$$\alpha = min[1,\ \text{the ratio of posterior probabilities of two models}]$$

which is analogous to the Metropolis-Hastings algorithm[10]. With the rule about moving to different models, the stochastic search eventually gives Markov chains which can show the 'walking' path through the model space during the search, and, more importantly, locate better fit-in models given the data.

By running MCMC long enough, the stochastic search can provide marginal posterior probabilities of $\beta_j \neq 0$ ($j \in [1, M]$), $p(\beta_j \neq 0|D)$, by averaging model parameter posterior probabilities[26] we can determine which one or ones are potential QTL.

3 ALGORITHM

3.1 Notation and Assumption

Phenotype matrix, denoted by Y, is the observation data matrix which stores quantitative trait data from the experiment. The element y_{ij} in the phenotypic matrix Y, gives the phenotypic value corresponding to the i^{th} genotype, also called line, and the j^{th} replicate. Also, in the phenotypic matrix Y, there should be at least 2 replicates for each genotype; however, it is not necessary to have the same number of replicates in each line. That is, for any i^{th} and j^{th} line($i \neq j$), in the phenotypic matrix Y data set, if there are n_i and n_j replicates, both n_i and n_j should be at least 2, but n_i may or may not be same as n_j. Considering the hierarchical model built in Section 2, all replicates are normally distributed with means and variances dependent on the line information, that is,

$$y_{ij} \sim N(\theta_i, \sigma_i^2)$$

The explanatory data matrices, known as the marker origin information matrices, denoted by X_m. Each one of the matrices is a subset of the Full marker information matrix, X_F, which contains all candidate marker origin information in the experiment. Unlike the general binary case, we use -0.5 or 0.5 to record marker origin information which represents whether the marker is from parent I or parent II in the marker origin information matrices.

In addition, in order to generate a Gibbs sequence and obtain samples from the stationary posterior distribution, we assume that parameters are initialized as following

- $\theta_i^{(0)}$: sample average of observed data in the i^{th} line

- $\sigma_i^{2(0)}$: sample variance of observed data in the i^{th} line
- $\tau^{2(0)}$: variance between sample means
- $\vec{\beta}^{(0)}$: estimates from a regression model based on the marker origin information matrix.

These estimates, along with the full conditional posterior distributions in Section 2, create samples from the posterior distribution. Due to the large dimensionality of the problem, we use 100,000 samples after a burn-in period of 2,000 samples[9, 11].

3.2 Model Comparison

Given the quantitative trait data Y, finding which model or models are better than the others in the model space is one of the most essential questions for QTL detection. Considering M candidate markers, the more likely the $q^{th}(q \leq 2^M)$ linear model fit the data set, the more potential marker or markers the model includes. That is, mathematically, it is important to find out the posterior probability of the q^{th} model among the whole model space, given the data D. The marginal posterior probability $P(k_q|D)$, called Posterior Model Probability[12], is the criterion for model comparison. By Bayes' rule, the Posterior Model Probability for the q^{th} model $P(k_q|D)$ can be derived from

$$P(k_q|D) = \frac{P(D|k_q) \times P(k_q)}{\sum_{i=1}^{|K|} P(D|k_i) \times P(k_i)} \tag{6}$$

where $|K|$ is the number of models in the model space. Since, for any model in the model space, we assume each of them is equally likely to be chosen, that is, $P(k_r) = P(k_s)$, for all $r, s \in |K|$. Thus, it is necessary to calculate $P(D|k_q)$, which is the probability of observing the data given the q^{th} model. This quantity can be

calculated by using the following integral,

$$P(D|k_q) = \int P(D|k_q, \vec{\Psi}_q) P(\vec{\Psi}_q|k_q) d\vec{\Psi} \tag{7}$$

where Ψ_q is the vector of parameters involved on the the q_{th} model. This quantity turns out to be computationally intensive[10, 12].

Because it is computationally difficult, the probability $P(D|k_q)$ can be approximated by Monte Carlo integration. Monte Carlo integration decomposes original function into the product of a function of x and probability density function $p(x)$, then expresses the original integral as an expectation of the function of x over the density $p(x)$. Thus, with a large number of random variables from the density function $p(x)$, the original integral can be approximated by averaging the sum of the function of random variables[22]. The Gibbs Sampler provides samples from the joint posterior distribution used in Equation (7). Therefore, the quantity $P(D|k_q)$ could be approximated by averaging the joint posterior probabilities produced by the Gibbs Sampler procedure[12], that is,

$$\int p(D|k_q, \vec{\Psi}_q) p(\vec{\Psi}_q|k_q) d\vec{\Psi} \approx \frac{1}{t} \sum_{j=m}^{T_G} p(D|k_q, \vec{\Psi}_q^{(j)}) \tag{8}$$

where j is index of time in Gibbs Sampler procedure, m is the index of the first time after burn-in period in the Gibbs Sampler procedure, T_G is the index of the ending time for the Gibbs Sampler, and $\vec{\Psi}_q^{(j)}$ is the vector of parameters in the j^{th} time of the Gibbs Sampler procedure for the q^{th} model.

3.3 Model Search by Stochastic Process

Besides the model comparison criterion, it is also very important to define some practical and effective strategies to search randomly through a subset of all possible models in the large model space for QTL detection research. For any candidate marker, it is either in the regression model or not. Therefore, there are 2^M possible models if M markers are considered as candidate QTL. Instead of the common searches, such as forward, backward, or stepwise regression, we use Monte Carlo Markov Chain Model Comparison, MC^3, a widely used stochastic search algorithm. We define the Model Selection Vector for the q^{th} model as $\vec{M}_q$. The length of $\vec{M}_q$, M is equal to the number of total candidate markers in the experiment, which is equivalent to the number of columns in the Full Marker Information Matrix X_F. Each s^{th} ($s \leq M$) element in $\vec{M}_q$ corresponds to its s^{th} marker counterpart. And the value of the s^{th} element is either 0 or 1, and defines whether the s^{th} marker is included in q^{th} model or not. For example, suppose there are 5 markers , L lines and n_i observations for each line. If $\vec{M}_q$ is [1,0,0,1,0], the q^{th} chosen model would include markers 1 and 4, that is,

$$\theta_i = \beta_0 + \beta_1 x_{1i} + \beta_4 x_{4i} + \epsilon_i.$$

where $i = 1, \ldots, L$.

The stochastic search first begins by randomly choosing $\vec{M}_q$, and calculating the Posterior Model Probability $p(k_q|D)$ of this model. Then, the algorithm randomly chooses a location along $\vec{M}_q$. At the chosen location, that value of the element is switched either from 0 to 1, or from 1 to 0. Referring the example above, if the vector $\vec{M}_q$ is [1,0,0,1,0], the $\vec{M}_{q+1}$ for the $(q+1)^{th}$ model could be [**0**,0,0,1,0], [1,**1**,0,1,0], [1,0,**1**,1,0], [1,0,0,**0**,0], or [1,0,0,1,**1**]. Say that the 3^{rd} location is chosen.

Then the new Model selection vector $\vec{M}_{q+1}$ is [1,0,1,1,0]. The new model would include marker 1, 3, and 4, and the Posterior model probability $p(k_{q+1}|D)$ is calculated for this model.

Therefore, during the model search procedure, it is possible to randomly search through different possible models in the huge model space by the Model Selection Vector $\vec{M}$ which is obtained randomly based on the previous one. Meanwhile, by using the Model Selection Vector $\vec{M}$, subsets of marker information for the each model can be chosen from Full Marker Information Matrix X_F as the predictor data.

3.4 Acceptance Probability

With Posterior Model Probability $P(k_q|D)$ for the q^{th} model, we can easily identify which model or models provide a better fit for the given quantitative trait data, and to locate potential markers associated with the quantitative trait. The posterior model probability can also be used to assist in the stochastic search through the model space. We create the ratio of the Posterior Model Probability of the new model to the one which is best so far, named Acceptance Probability. An **Acceptance Probability**, defined as $\alpha_{i+1,i}$, is the probability that the i^{th} model, which is best so far, should be replaced by the $(i+1)^{th}$ model. By Metropolis-Hasting algorithm, an Acceptance Probability $\alpha_{i+1,i}$ is defined as minimum between 1 and the ratio of Posterior Model Probabilities of the $(i+1)^{th}$ model to the i^{th} model which is the best-fit model given data[10, 11], that is,

$$\alpha_{i+1,i} = min\left[1, \frac{p(k_{i+1}|D)}{p(k_i|D)}\right] \tag{9}$$

The Acceptance Probability $\alpha_{i+1,i}$ is similar to transition probability in a Markov Chain, which is the probability that the chain moves from one state to another state. The chain may become stuck at a locally optimum model (i.e. transition probabilities approaches 0), therefore, it is important to have different chains. In general, 10 or more different chains and more than 1,000 steps for each chain are necessary[1].

The way our algorithm decides if the chain should move to the new model is by defining the acceptance probability as a success probability p of a Bernoulli trial. Suppose the i^{th} model is best so far and the new model $(i+1)^{th}$ model is the model in question, by using Acceptance Probability $\alpha_{i+1,i}$ as the success probability, we randomly generate a bernoulli random variable with success probability $\alpha_{i+1,i}$. If the number is 1, then the chain moves to the new model. That is, if this move occurs, the new model, the $(i+1)^{th}$ model, would replace the i^{th} model which is the best-fit model given the data. A tally for each best model is kept. This tally indicates the frequency in which each model is defined as the best-fit model.

3.5 Activation Probability

After recording this information for each chain, it becomes possible to locate which candidate marker or markers show the most potential as QTL. Given the quantitative trait data set, we calculate the Activation Probability for each marker by $p(\beta_j \neq 0|D)$, which is defined as

$$p(\beta_j \neq 0|D) = \sum_{i=1}^{|K|} p(\beta_j \neq 0|k_i, D)p(k_i|D) \tag{10}$$

where $|K|$ is the total length of all the chains, and k_q is the q^{th} model. Moreover, by Bayesian model averaging[23], $\beta_j \neq 0$ is dependent on whether the j^{th} marker is included in best models or not. That is, the Activation Probability of j^{th} candidate marker is the weighted frequency of the j^{th} marker showing up in the best models during the model search procedure. That is,

$$p(\beta_j \neq 0|D) = \sum_{i=1}^{|K|} I_j \times p(k_i|D), \text{ where } I_j = \begin{cases} 1, & \text{if the } j^{th} \text{ marker is in the } i^{th} \text{ model} \\ 0, & \text{otherwise} \end{cases} \tag{11}$$

4 SIMULATION

4.1 Data Set

We apply our new algorithm to a simulated data set. We assume that there is only one QTL which is located on the 4^{th} marker among a total of 10 candidate markers, and there are 20 different genotypes in a plant experiment. We randomly generate all markers origin information by 0.5 and -0.5 which indicates markers are either from parent A or parent B. The Full Marker Information Matrix X_F in the simulation is shown as Table 1.

For each genotype in the simulation, the number of the quantitative trait observations for i^{th} genotype, denoted by n_i, is randomly chosen between 2 and 15, shown on the 2^{nd} column of the phenotypic matrix Y(Table 2). Table 2 illustrates the first 4 genotypes and the last 3 genotypes with n_i = 14, 5, 15, 10, 3, 10, and 15, respectively. Any observation y_{ij} in the matrix Y, which is the i^{th} line, j^{th} replicate ($i \in [1, 2, \ldots, 20], j \in [1, \ldots, n_i]$, where n_i is the number of replicates of i^{th} line), is generated based on the i^{th} line, 4^{th} marker origin information, that is,

$$y_{ij} = 35 + 10 \times x_{i4} + \epsilon_{ij}, (i \in [1, 2, \ldots, 20], j \in [2, \ldots, n_i]) \tag{12}$$

where x_{i4} is the i^{th} line, 4^{th} marker information from X_F(Table 1), n_i is the number of replicates in the i^{th} line, and ϵ_{ij} is random draw from the standard normal distribution, which is a normal with mean 0 and standard deviation 1.

Full Marker Information Matrix X_F										
Genotype	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}
1	0.5	-0.5	-0.5	-0.5	-0.5	-0.5	-0.5	0.5	-0.5	-0.5
2	-0.5	0.5	-0.5	0.5	0.5	0.5	-0.5	0.5	0.5	-0.5
3	0.5	-0.5	0.5	0.5	0.5	-0.5	0.5	-0.5	0.5	0.5
4	0.5	-0.5	-0.5	0.5	-0.5	-0.5	0.5	0.5	0.5	-0.5
5	-0.5	0.5	0.5	-0.5	0.5	0.5	-0.5	-0.5	-0.5	0.5
6	0.5	0.5	0.5	-0.5	0.5	-0.5	-0.5	-0.5	-0.5	0.5
7	0.5	-0.5	-0.5	-0.5	0.5	0.5	0.5	0.5	-0.5	0.5
8	0.5	-0.5	0.5	0.5	0.5	0.5	-0.5	0.5	0.5	-0.5
9	0.5	-0.5	-0.5	-0.5	0.5	0.5	0.5	-0.5	-0.5	0.5
10	0.5	-0.5	-0.5	0.5	0.5	-0.5	0.5	0.5	0.5	-0.5
11	-0.5	0.5	0.5	-0.5	-0.5	0.5	-0.5	-0.5	0.5	0.5
12	-0.5	-0.5	0.5	0.5	-0.5	0.5	-0.5	0.5	-0.5	0.5
13	-0.5	0.5	0.5	-0.5	0.5	0.5	-0.5	0.5	-0.5	0.5
14	-0.5	0.5	-0.5	0.5	-0.5	0.5	0.5	0.5	-0.5	-0.5
15	0.5	-0.5	-0.5	0.5	-0.5	0.5	-0.5	0.5	-0.5	0.5
16	-0.5	-0.5	0.5	0.5	-0.5	-0.5	-0.5	0.5	-0.5	-0.5
17	-0.5	-0.5	0.5	0.5	0.5	-0.5	-0.5	-0.5	0.5	-0.5
18	-0.5	-0.5	-0.5	-0.5	0.5	0.5	-0.5	-0.5	-0.5	-0.5
19	-0.5	-0.5	-0.5	0.5	0.5	-0.5	-0.5	0.5	-0.5	-0.5
20	-0.5	0.5	0.5	-0.5	-0.5	-0.5	-0.5	-0.5	0.5	-0.5

Table 1: Full marker origin information matrix

Phenotypic Matrix Y									
	n_i	*Observation 01-08*							
1	14	29.5674	30.2944	28.3959	30.0000	30.6232	28.8122	30.1286	29.6694
2	5	38.3344	38.6638	40.2573	39.6821	40.7990			
3	15	40.1253	40.7143	38.9435	41.0950	40.9409	40.9863	38.8322	40.4978
4	10	40.2877	41.6236	41.4151	38.1260	39.0079	39.4814	39.5394	41.4885
⋮									
18	3	30.0593	28.7975	28.9909					
19	10	39.9044	39.9802	39.9805	41.4435	39.1783	39.7389	38.7081	39.3935
20	15	29.1677	29.8433	29.9518	29.6490	29.7344	30.9535	29.9271	28.6526

	Observation 09-15								
1	14	30.4694	29.5350	30.6353	28.9819	30.4855	29.4588		
2	5								
3	15	40.0359	40.7283	40.5512	41.5210	39.7238	41.0727	38.4825	
4	10	39.3725	42.1122						
⋮									
18	3								
19	10	40.0558	41.1902						
20	15	29.6321	28.8838	29.2957	28.8929	30.2809	29.8839	31.2698	

Table 2: Quantitative trait matrix

4.2 Simulation

Using the method described previously, we calculated posterior model probabilities by running the Gibbs Sampler 102,000 times for each model with a burn-in of 2,000. The remaining 100,000 samples are used to estimate the posterior model probability.

To avoid incidences where a chain becomes stuck at a local optimum model during model search, there are a total of 50 chains for the simulation, and each one of them has a different starting model which is defined as the best model at the very beginning of the chain. For each chain, we randomly search 2,000 different possible models based on the Model Selection Vector $\vec{M}$ and Acceptance Probability. During the whole search and comparison procedure, we record the number in which each model is defined as the best model. Afterwards, we calculate the weighted frequency, or activation probability, of each marker included in the different best model. Any markers with activation probability greater than 0.5 is identified as a potential QTL.

4.3 Output

The program is written in MatLab (R) 2007b (V7.5.0.342) and executed on Intel(R) Core(TM)2 CPU 6400 @ 2.13GHz, 1.00GB of RAM PC with the operation system Microsoft Windows XP Professional SP2. The simulation takes approximately 17 hours. The output from the algorithm is shown in **Model Probability** (Table 3), **Best-Fit Models Chain** (Table 4), and **Marker Activation Probability** (Table 5).

In the table Model Probability (Table 3), each column corresponds(from left to right) to **Model Index**[1],Posterior Model Probability, and the number of models defined as the best model. By comparing the last 2 columns in the Model Probability table, we could determine which model or models are a better-fit to the hierarchical linear model for the given data, which are best-fit models in the simulation. In the simulation, the model 217, which is the regression model including markers 3, 4, 6, 7, and 10, is one of the best models with the highest posterior probability. And the model 332, which has markers 2, 4, 7, and 8 in the model, is one with the highest frequency as the best model during stochastic search.

In the table Best-Fit Model Chains (Table 4), each column shows the "best-fit" model path of each chain by using model index. It shows either the chain stays with the current "best-fit" model or replaces it by the new model. According to the table Best-Fit Model Chains, we could visualize that, in the different chains, the best models move among the models which are a better fit to the hierarchical model given the data shown in Table 3.

The table Marker Activation Probability (Table 5) includes a list of models sorted by the number of times when each model is chosen as the best model. For each model in the table, the first 10 columns are the Model Selection Vector $\vec{M}$, and the 11^{th} column is the number of the model when it is defined as the best model during the whole search and comparison procedure. The Activation Probabilities are shown at the last line of the table. Considering all candidate markers, we treat the markers with Activation probabilities more than 50% as the QTL[8, 10, 12]. Besides, it is apparent that the models which include the 4^{th} marker and 8^{th} are much more frequently defined as the best model than the others in the simulation. Therefore, they

[1]the Model Selection Vector value converted from binary to decimal

Index	Posterior Model Probability	Frequency
0	0.0000	26
1	0.0000	30
2	0.0000	18
3	0.0000	33
	⋮	
217	0.090972	254
	⋮	
332	0.041887	272
	⋮	
1022	0.0002	95
1023	0.0000	69

Table 3: Model Probability

Time	Chain 1	Chain 2	...	Chain 49	Chain 50
1	407	411	...	775	584
2	407	410	...	791	586
3	471	411	...	791	842
			⋮		
999	215	418	...	872	71
1000	211	386	...	876	70
1001	431	407	...	876	70
			⋮		
1998	221	934	...	334	580
1999	220	950	...	335	580
2000	216	438	...	463	580

Table 4: Best Model Chains

Model Selection Vector										Freq
X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}	
0	1	0	1	0	0	1	1	0	0	272
1	1	0	1	1	1	0	0	0	1	263
0	0	0	1	0	1	1	0	0	1	255
⋮										
1	1	1	0	0	0	0	1	1	1	65
0	0	0	0	0	1	0	1	0	0	65
⋮										
1	0	1	0	1	0	1	0	0	0	5
1	0	1	0	0	0	1	0	0	0	4
1	0	1	0	1	1	0	0	0	1	3
Activation Probability										
0.4590	0.4952	0.4604	0.7818	0.4582	0.4834	0.4668	0.5106	0.4759	0.4922	

Table 5: Marker Activation Probability

should be considered as the QTL among the all candidate markers. Furthermore, by comparing the marker origin information in the Full marker information matrix X_F, we notice that there is 75% similar between 4^{th} marker's origin information and 8^{th} markers's. And, the 4^{th} marker's Activation Probabilities is much higher than the other's, on the contrary, the 8^{th}'s is very close to 50%. More importantly, Activation Probabilities for true QTL should be much higher than markers that are not associated with the quantitative trait.

5 CONCLUSION AND DISCUSSION

In this research, we have shown how to use a stochastic search algorithm to search through a model space by using MC^3 with the Acceptance probabilities in a Bayesian hierarchical regression setting. We approximate each model posterior model probability by using Monte Carlo integration with samples generated via the Gibbs Sampler algorithm. More importantly, with the idea of Bayesian model averaging, we are able to locate which locus or loci are more important in predicting a quantitative trait by using Activation Probabilities, which is the weighted frequency of each marker in the best models during the whole stochastic search. By using Bayesian hierarchical modeling, our algorithm has shown to be very effective for the QTL detection problem. In addition, unlike the other methods which require one observation per genotype, our algorithm can handle multiple replicates in each genotype, and the situation where the number of replicates in different genotypes are different, which makes the algorithm more flexible.

We ran a simulation study for the algorithm, and the results satisfy our assumption. Also, we identify some issues which could be considered as plans for improvement.

- Since there are 2^M possible regression models when there are M candidate markers, the number of possible models becomes very large which is definitely an issue for computing. Clustering may be a good idea in this situation, that is, group markers and try to restrict the analysis to regions with potential QTL.

- For the ideal simulation data set, we update all posterior distribution by Gibbs Sampler only 102,000 times, and consider first 2,000 times as the burn-in periods. However, it might not be long enough for the real data. Therefore, we may apply Gibbs Sampler to draw 500,000 samples and throw out first 100,000.

- In addition, it will cost much more computing time when there are plenty of different genotypes from the real experiment. In this case, MatLab may not be the best choice for the whole algorithm. We plan to make use of MatLab advantages in matrices and vectors, and compute different probabilities by other language, such as FORTRAN, which has abundant packages for different distributions and is more efficient for simulation.

- There are a lot of studies showing the importance of interactions between QTL. The next step for this research is incorporating interactions in the QTL model.

REFERENCES

[1] Broman K.W., A model selection approach for the identification of quantitative trait loci in experimental crosses. *Journal of Royal Statistical Society.* 64, Part 4, pp.641-656, 2002.

[2] Lander E.S, Botsterin D., Mapping Mendalian factors underlying quantitative traits using RFLP linkage maps. *Genetics* 121:185-199, 1989.

[3] Berry CC., Computationally Efficient Bayesian QTL Mapping in Experimental Crosses. *ASA Proceedings of the Biometrics Section*, pp.164-169, 1998.

[4] Broman KW, Wu H, Sen S, QTL Mapping in experimental crosses. *Bioinformatics*,19:889-890, 2003.

[5] Gelman A., Carlin A.J, Stern H.S and Rubin D.B., *Bayesian Data Analysis*, 2nd edn, Chapman Hall/CRC, Boca Raton London New York Washington, D.C., 2004.

[6] Broman K.W, The Genomes of Recombinant Inbred Lines. *Genetics Society of America*, DOI:10.1534/genetics.104.035212, 2004

[7] Loudet O, Chaillou S, Daniel-Vedele F, Bay-0 × Shahdara recombination inbred line population: a powerful tool for the genetic dissection of complex traits in Arabidopsis, *Theoretical and Applied Genetics,* Vol. 104, 1173-1184, 2002.

[8] Boone E.L , Ye K, Smith E.P, Evaluating the Relationship Between Ecological and Habitat Conditions Using Hierarchical Models, *Journal of Agriculture, Biological, and Environmental Statistics*, Vol. 10, Number 2 Page 1-17, 2005

[9] Berg B.A, Markov Chain Monte Carlo Simulations and Their Statistical Analysism. *World Scientific* ISBN 981-238-935-0, 2004.

[10] Boone E.L, Simmons S.J, Ye K, Stapleton A.E, Analyzing Quantitative Trait Loci for the Arabidopsis thaliana using Markov Chain Monte Carlo Model Composition with restricted and unrestricted model spaces. *Statistical Methodology* 3 (2006) 69-78

[11] Walsh B., Markov Chain Monte Carlo and Gibbs Sampling. *Lecture Note for EEB* 581, version 26, April, 2004.

[12] Bao H, *Bayesian Hierarchical regression model to detect Quantitative trait loci.* UNCW, 2006

[13] Lee P.M, *Bayesian statistics: an introduction*, Edward Arnold, A division of Hodder and Stoughton, LONDON MELBOURNE AUCKLAND, 1989

[14] Congdon P, *Applied Bayesian Modelling*, John Wiley and Sons Ltd, 2003

[15] Lynch M, Walsh B, *Genetics and Analysis of Quantitative Traits*, Sinauer Associates, Inc., Sunderland, MA, 1998.

[16] Jansen R.C, Interval mapping of multiple quantiative trait loci. *Genetics*; **135**:205-11,1993.

[17] Jansen R.C. and Stam P., High resolution of quantitative traits into multiple loci via interval mapping. *Genetics*; **136**:1447-55,1994.

[18] Zeng Z.B., Theoretical basis for separation of multiple linked gene effects in mapping quantitative trait loci. *Proc. Natl. Acad. Sci. USA*; **90**:10972-6,1993.

[19] Zeng Z.B., Precision mapping of quantitative trait loci *Genetics*; **136**:1457-68,1994.

[20] Broman K.W., Review of statistical methods for QTL mapping in experimental crosses *Lab Animal*; **30(7)**:44-52, 2001.

[21] Simmons S.J. and Stapleton A.E, Bayesian hierarchical models to detect quantitative trait loci *Chance*; **VOL.19,NO.3**:11-14, 2006.

[22] Walsh B., Markov Chain Monte Carlo and Gibbs Sampling, *Lecture Notes for EEB*; 581:Version 26, 2004.

[23] Congdon P, *Bayesian Statistical Modelling*, 2nd Edition, John Wiley & Son. Ltd.

[24] Michael Lavine, ISDS, Duke University, Durham, North Carolina, *What is Bayesian statistics and why everything else is wrong*

[25] Younan Chen, Keying Ye, Department of Statistics, Virginia Tech, Blackburg, *A Bayesian Hierarchical Approach to Dual Response Surface Modelling.*

[26] David B. Dunson, Practical Advantages of Bayesian Analysis of Epidemiologic, *American Journal of Epidemiology*; **VOL. 153. No. 12**:1222-6, 2001

[27] Cystic Fibrosis Mutation Database. Cystic Fibrosis Consortium Web site. Available at http://www.genet.sickkids.on.ca/cftr. Accessed May 12, 2005.

[28] Hedden P., Green Revolution Genes, Rothamsted Research, Harpenden, Hertfordshire, UK, *Plant Physiology*; **Chapter20:Essay20.2**, August, 2006.

[29] Lynch M, Walsh B. *Genetics and Analysis of Quantitative Traits*, December 1997.

APPENDIX

A. QTL detection from stochastic process Mathlab code

```
%Parameter Beta
function [ vecOutBetaInit ] = funcBetaInit(matxArgMarker, . . .
matxArgRgrnResp, matxArgRespMS)
   matxRgrnMarker=funcRgrnMarker(matxArgMarker,matxArgRespMS);
vecOutBetaInit=inv(matxRgrnMarker'*matxRgrnMarker) . . .
*(matxRgrnMarker')*matxArgRgrnResp;

function [ vecOutBetaUdtMiu, matxOutBetaUdtCov ] = funcBetaUdtParas( . . .
matxArgRgrnSlt, vecArgTheta, doubleArgThetaVar, doubleArgBetaVar)
    matxMTM=(matxArgRgrnSlt'*matxArgRgrnSlt) . . .
/doubleArgThetaVar;
    matxSizeMTM=size(matxMTM);
    matxMTM=matxMTM+(speye(matxSizeMTM(1,1))/doubleArgBetaVar);

    matxOutBetaUdtCov=inv(matxMTM);
    vecOutBetaUdtMiu=matxOutBetaUdtCov . . .
*((matxArgRgrnSlt'*vecArgTheta)/doubleArgThetaVar);

function [ vecOutBetaUdt ] = funcBetaUdt( matxArgRgrnSlt, . . .
vecArgTheta, doubleArgThetaVar, doubleArgBetaVar)
    [vecBetaUdtMiu, matxBetaUdtCov]= funcBetaUdtParas(matxArgRgrnSlt, . . .
vecArgTheta, doubleArgThetaVar, doubleArgBetaVar);
    matxBetaUdtCovCHFAC=chol(matxBetaUdtCov);
    vecSizeBeta=size(vecBetaUdtMiu);
```

```
vecOutBetaUdt=matxBetaUdtCovCHFAC*randn(vecSizeBeta(1,1),1)+vecBetaUdtMiu;

%Parameter Sigma
function vecOutStdGammaRnd=funcStdGammaRnd(vecGammaAlpha,intArgNoLine)
    vecGammaAlphaFlr=floor(vecGammaAlpha);
    vecGammaAlphaRmd=vecGammaAlpha-vecGammaAlphaFlr;
    vecOutStdGammaRnd=(vecGammaAlphaRmd>0).*(gamma(1+ . . .
vecGammaAlphaRmd).*rand(intArgNoLine,1).^(1 . . .
./(vecGammaAlphaRmd+(vecGammaAlphaRmd==0))));
    for i=1:intArgNoLine
        vecOutStdGammaRnd(i,1)=vecOutStdGammaRnd(i,1). . .
+sum(-log(rand(vecGammaAlphaFlr(i,1),1)));
    end

function [ vecOutSigmaAlpha ] = funcSigmaAlpha( matxArgRespMS, . . .
intArgNoLine, doubleArgSigmaNull )
vecOutSigmaAlpha=ones(intArgNoLine,1)*doubleArgSigmaNull. . .
+(matxArgRespMS(:,1)/2);

function [vecOutSigmaInit]=funcSigmaInit(matxArgRespMS,intArgNoLine)
    %add one more argument: intArgNoLine
    %change the argument: matxArgResp by matxArgRespMS;
    if(sum(matxArgRespMS(:,1)==1)==0)
        vecOutSigmaInit=(matxArgRespMS(:,3)-((matxArgRespMS(:,2).^2). . .
./matxArgRespMS(:,1))./(matxArgRespMS(:,1)-1);
    else
        vecOutSigmaInit=zeros(intArgNoLine,1);
```

```
        for rowLp=1:1:intArgNoLine
            if(matxArgRespMS(rowLp,1)==1)
                vecOutSigmaInit(rowLp,1)=1;
            else
                vecOutSigmaInit(rowLp,1)=(matxArgRespMS(rowLp,3) . . .
-((matxArgRespMS(rowLp,2)^2)/matxArgRespMS(rowLp,1))). . .
/(matxArgRespMS(rowLp,1)-1);
            end
        end
    end

function [ vecOutSigmaUdtNum ] = funcSigmaUdtNum(matxArgRespMS,vecArgTheta)
vecOutSigmaUdtNum=0.5*(1+(matxArgRespMS(:,3)-(2*vecArgTheta. . .
.*matxArgRespMS(:,2))+(matxArgRespMS(:,1).*(vecArgTheta.^2))));

function [ vecOutSigmaUdt ] = funcSigmaUdt(matxArgRespMS,. . .
intArgNoLine, vecArgTheta, vecArgSigmaAlpha )
    vecSigmaUdtNum=funcSigmaUdtNum(matxArgRespMS, vecArgTheta);
    vecSigmaUdtDen=funcStdGammaRnd(vecArgSigmaAlpha,intArgNoLine);
vecOutSigmaUdt=vecSigmaUdtNum./vecSigmaUdtDen;

%Parameter Theta
function [ vecOutThetaInit, matxOutRespMS ] = funcThetaInit( matxArgResp )
matxOutRespMS=funcSampleMS(matxArgResp);
vecOutThetaInit=matxOutRespMS(:,2)./matxOutRespMS(:,1);

function [ vecOutThetaUdtMiu, vecOutThetaUdtSD ] = . . .
```

```
funcThetaUdtParas(matxArgRgrnSlt, matxArgRespMS,...
vecArgBeta, doubleArgThetaVar, vecArgSigma)
    vecRespEst=matxArgRgrnSlt*vecArgBeta;
vecOutThetaUdtSD=realsqrt((doubleArgThetaVar*vecArgSigma). . .
./(matxArgRespMS(:,1)*doubleArgThetaVar+vecArgSigma));
vecOutThetaUdtMiu=(vecOutThetaUdtSD.^2). . .
.*(vecRespEst/doubleArgThetaVar+matxArgRespMS(:,2)./vecArgSigma);

function [ vecOutThetaUdt ] = funcThetaUdt(matxArgRgrnSlt, . . .
matxArgRespMS, intArgNoLine, vecArgBeta, doubleArgThetaVar, vecArgSigma)
    [vecThetaUdtMiu, vecThetaUdtSD ]=. . .
funcThetaUdtParas(matxArgRgrnSlt, matxArgRespMS, . . .
vecArgBeta, doubleArgThetaVar, vecArgSigma);
vecOutThetaUdt=vecThetaUdtMiu+vecThetaUdtSD.*randn(intArgNoLine,1);

%Parameter Tau
function [doubleOutThetaVarInit]=funcThetaVarInit(vecArgTheta)
    matxThetaMS=funcSampleMS(vecArgTheta');
    doubleOutThetaVarInit=(matxThetaMS(1,3). . .
-((matxThetaMS(1,2)^2)/matxThetaMS(1,1)))/(matxThetaMS(1,1)-1);

function [doubleOutThetaVarNum]=funcThetaVarUdtNum(matxArgRgrnSlt,. . .
intArgNoLine, vecArgTheta, vecArgBeta)
matxResidual=vecArgTheta-matxArgRgrnSlt*vecArgBeta;
doubleOutThetaVarNum=(matxResidual'*matxResidual+intArgNoLine)/2;

function [ doubleThetaVar ] = funcThetaVarUdt(matxArgRgrnSlt,. . .
```

```
intArgNoLine, vecArgTheta, vecArgBeta, doubleArgThetaVarAlpha)
doubleThetaVarNum=funcThetaVarUdtNum(matxArgRgrnSlt,. . .
intArgNoLine, vecArgTheta, vecArgBeta);
doubleThetaVarDen=funcStdGammaRnd(doubleArgThetaVarAlpha, 1);
    doubleThetaVar=doubleThetaVarNum/doubleThetaVarDen;

%Likelihood function
function [ doubleOutLikelihood, indicOutCounter ] = . . .
funcLikelihood( matxArgRgrnSlt, matxArgRespMS, ...
vecArgTheta, vecArgBeta, doubleArgThetaVar, vecArgSigma, ...
doubleArgThetaVarAlpha, vecArgSigmaAlpha, doubleArgBetaVar, doubleAdjLE)
    vecEstMarker=matxArgRgrnSlt*vecArgBeta;

    doubleLikeliPart1=0-sum(log(vecArgSigma). . .
.*vecArgSigmaAlpha);
    doubleLikeliPart1=doubleLikeliPart1. . .
-sum((1./(2*vecArgSigma)));
    doubleLikeliPart1=doubleLikeliPart1. . .
-sum(((vecArgTheta-vecEstMarker).^2)/(2*doubleArgThetaVar));

    vecResidual=matxArgRespMS(:,3). . .
-(2*matxArgRespMS(:,2).*vecArgTheta). . .
+(matxArgRespMS(:,1).*(vecArgTheta.^2));
    vecResidual=vecResidual./(2*vecArgSigma);
    doubleLikeliPart2=0-sum(vecResidual);

    doubleOutLikelihood=doubleLikeliPart1+doubleLikeliPart2;
```

```
    doubleOutLikelihood=doubleOutLikelihood. . .
-(doubleArgThetaVarAlpha*log(doubleArgThetaVar));
    doubleOutLikelihood=doubleOutLikelihood. . .
-(1/(2*doubleArgThetaVar));
    doubleOutLikelihood=doubleOutLikelihood. . .
-((1/(2*doubleArgBetaVar))*(vecArgBeta'*vecArgBeta));

    doubleOutLikelihood=doubleOutLikelihood+abs(doubleAdjLE);

    if doubleOutLikelihood>10
        indicOutCounter=1;
        doubleOutLikelihood=-999999.000000000000;
    else
        indicOutCounter=0;
    end

%Gibbs Sampler
function [doubleOutLLike] =funcLLike(matxArgModelSlt,. . .
matxArgMarkerFull, matxArgRgrnResp, matxArgRespMS, . . .
intArgNoLine, intArgNoMarker, intNoTimesGS, intNoBP, . . .
oubleArgInitThetaVar, vecArgInitSigma, . . .
doubleArgThetaVarAlpha, vecArgSigmaAlpha, doubleArgBetaSigma, . . .
doubleArgAdjLE, intArgFuncChoose)

    matxMarkerSlt=funcMarkerSelect(matxArgMarkerFull, . . .
matxArgModelSlt, intArgNoMarker);
    if(isempty(matxMarkerSlt)~=1)
```

```
        matxRgrnSlt=ones(intArgNoLine,1);
        matxRgrnSlt=[matxRgrnSlt,matxMarkerSlt];
    else
        matxRgrnSlt=ones(intArgNoLine,1);
    end

    vecBeta=funcBetaInit(matxMarkerSlt, matxArgRgrnResp, matxArgRespMS);
    doubleThetaVar=doubleArgInitThetaVar;
    vecSigma=vecArgInitSigma;

    switch(intArgFuncChoose)
        case 0
            doubleMLE=-999999999999999999;
            for t=1:1:intNoTimesGS
                vecTheta=funcThetaUdt(matxRgrnSlt, matxArgRespMS, . . .
intArgNoLine, vecBeta, doubleThetaVar, vecSigma);
                doubleThetaVar=funcThetaVarUdt(matxRgrnSlt, . . .
intArgNoLine, vecTheta, vecBeta, doubleArgThetaVarAlpha);
                vecBeta=funcBetaUdt(matxRgrnSlt, vecTheta, . . .
doubleThetaVar,doubleArgBetaSigma);
                vecSigma=funcSigmaUdt(matxArgRespMS, . . .
intArgNoLine, vecTheta, vecArgSigmaAlpha);

                if(t>intNoBP)
                    [ doubleLLE, intCounter ] = . . .
funcLikelihood( matxRgrnSlt, matxArgRespMS, vecTheta, . . .
vecBeta, doubleThetaVar, vecSigma, . . .
```

```
doubleArgThetaVarAlpha, vecArgSigmaAlpha, . . .
doubleArgBetaSigma, doubleArgAdjLE);
                    if(intCounter~=1 && doubleLLE>doubleMLE)
                        doubleMLE=doubleLLE;
                    end
                end
            end
            doubleOutLLike=doubleMLE;
        case 1
            doubleAvgLE=0;
            for t=1:1:intNoTimesGS
                vecTheta=funcThetaUdt(matxRgrnSlt, . . .
matxArgRespMS, intArgNoLine, vecBeta, . . .
doubleThetaVar, vecSigma);
                doubleThetaVar=funcThetaVarUdt(matxRgrnSlt, . . .
intArgNoLine,vecTheta, vecBeta, doubleArgThetaVarAlpha);
                vecBeta=funcBetaUdt(matxRgrnSlt, vecTheta, . . .
doubleThetaVar, doubleArgBetaSigma);
                vecSigma=funcSigmaUdt(matxArgRespMS, . . .
intArgNoLine, vecTheta, vecArgSigmaAlpha);

                if(t>intNoBP)
                    [ doubleLLE, intCounter ] = . . .
funcLikelihood( matxRgrnSlt, matxArgRespMS, vecTheta,  . . .
vecBeta, doubleThetaVar, vecSigma, doubleArgThetaVarAlpha, . . .
vecArgSigmaAlpha, doubleArgBetaSigma, doubleArgAdjLE);
                    if(intCounter~=1)
```

```
                        doubleAvgLE=doubleAvgLE+exp(doubleLLE);
                    end
                end
            end
            doubleOutLLike=doubleAvgLE/(intNoTimesGS-intNoBP)
        otherwise
            disp 'Error';
    end

%Model search
function ranInt = funcRandInt(outputRow,outputCol,outputRange,varargin)

if isequal(size(outputRange),[1 2]) . . .
&& ~isequal(outputRange(1),outputRange(2)-1),
    warning('To specify a range [low high] use [low:high].')
end
if ~isequal(round(outputRange),outputRange),
    warning('Specified RANGE contains noninteger values.')
end
if ~isequal(length(outputRange),length(outputRange(:))),
    error('Range must be a vector of integer values.')
end

numElements = outputRow*outputCol;

if isempty(varargin),
```

```
    ranInt = zeros(outputRow,outputCol);
    randIx = floor((length(outputRange))*rand(size(ranInt))) + 1;
    ranInt = outputRange(randIx);
    if ~isequal(size(randIx),size(ranInt)),
        ranInt = reshape(ranInt,size(randIx));
    end

elseif isequal(varargin{1},'noreplace'),

    if numElements > length(outputRange),
        error('Not enough elements in range to . . .
sample without replacement.')
    else
        % Generate full range of integers
        XfullShuffle = outputRange(randperm(length(outputRange)));
        % Select the first bunch:
        ranInt = reshape(XfullShuffle(1:numElements),outputRow,outputCol);
    end

else
    error('Valid argument is ''noreplace''.')
end

function [matxOutMarkerSelected]=. . .
funcMarkerSelect(matxArgMarkerFull,matxArgModelSlted,intArgNoMarker)
matxOutMarkerSelected=[];
for i=1:1:intArgNoMarker
```

```
    if(matxArgModelSlted(1,i)==1)
        matxOutMarkerSelected=[matxOutMarkerSelected, . . .
matxArgMarkerFull(:,i)];
    end
end

function [matxOutModelFull]=funcModelFull(intArgNoMarker, intArgNoLine)

    intNoRgrnVar=intArgNoMarker+1;

    intTimesML=intNoRgrnVar/intArgNoLine;

    if(intNoRgrnVar<=intArgNoLine)
        matxOutModelFull=ones(1,intArgNoMarker);
    else
        matxOutModelFull=zeros(1,intArgNoMarker);
        if(((fix(intTimesML)-1)*intArgNoLine) . . .
< intNoRgrnVar <= (fix(intTimesML)*intArgNoLine))
            intNoZeros=fix(intTimesML);
            matxOutModelFull(1:intNoZeros+1:intArgNoMarker)=1;
        end
    end

function [intOutModelValue, matxOutModelSlted] = . . .
funcModelSelect( intArgNoMarker )

    intModelVal=0;
```

```
    matxModelSelected=zeros(1,intArgNoMarker);
    for j=1:1:intArgNoMarker % j: the power for every column;
        doublePro=rand;
        if(rand(1)<doublePro)
            matxModelSelected(1,j)=1;
        else
            matxModelSelected(1,j)=0;
        end
        weight=2^(intArgNoMarker-j); % the weight of every column;
        if(matxModelSelected(1,j)==0)
            continue;
        else
            intModelVal=intModelVal+matxModelSelected(1,j)*weight;
        end
    end
intOutModelValue=intModelVal;
matxOutModelSlted=matxModelSelected;

function [intOutModelVal,matxOutModelSwitch]=. . .
funcModelSwitch(intArgModelVal, matxArgModelSltOrg)

    matxSizeModelSltOrg=size(matxArgModelSltOrg);
    intNoMarkers=matxSizeModelSltOrg(1,2);
    if(matxSizeModelSltOrg(1,1)~=1)
    end
```

```
    intModelVal=0;
    intIndexSwitch=funcRandInt(1,1,[1:intNoMarkers]);
    switch matxArgModelSltOrg(1,intIndexSwitch)
        case 0
            intModelVal=intArgModelVal+2^. . .
(intNoMarkers-intIndexSwitch);
            matxArgModelSltOrg(1,intIndexSwitch)=1;
        case 1
            intModelVal=intArgModelVal-2^. . .
(intNoMarkers-intIndexSwitch);
            matxArgModelSltOrg(1,intIndexSwitch)=0;
        otherwise
            disp('Error');
    end

matxOutModelSwitch=matxArgModelSltOrg;
intOutModelVal=intModelVal;

%Transform X, Y to appropriate format
function [ matxOutRgrnMarker ] = . . .
funcRgrnMarker(matxArgMarker, matxArgRespMS)
    intNoRep=sum(matxArgRespMS(:,1));

    if(isempty(matxArgMarker))
        matxOutRgrnMarker=ones(intNoRep,1);
    else
        vecSizeMarker=size(matxArgMarker);
```

```
        matxOutRgrnMarker=zeros(intNoRep,vecSizeMarker(1,2)+1);
        intIndexRow=1;

        for rowMarker=1:1:vecSizeMarker(1,1)
            for times=1:1:matxArgRespMS(rowMarker,1);
                matxOutRgrnMarker(intIndexRow,:)=. . .
[1,matxArgMarker(rowMarker,:)];
                intIndexRow=intIndexRow+1;
            end
        end
    end

function [ vecOutRgrnResp ] = funcRgrnResp( matxArgResp, . . .
matxArgRespMS, intArgNoLine )
%matxRespMS=funcSampleMS(matxArgResp);
%matxSizeRespMS=size(matxRespMS);
vecOutRgrnResp=[];
for rows=1:intArgNoLine
    intIndexCol=1;
    for cols=1:1:matxArgRespMS(rows,1)
            vecOutRgrnResp=[vecOutRgrnResp;. . .
matxArgResp(rows,intIndexCol)];
            intIndexCol=intIndexCol+1;
    end
end

function [ matxOutSampleMS ] = funcSampleMS( matxParaMatx )
```

```
matxSizeParaMatx=size(matxParaMatx);
matxSampleMS=zeros(matxSizeParaMatx(1,1),3);
for rowParaMatx=1:1:matxSizeParaMatx(1,1)
    n=0;
    ysum=0;
    y2sum=0;
    for colParaMatx=1:1:matxSizeParaMatx(1,2)
        if(isnan(matxParaMatx(rowParaMatx,colParaMatx))~=1)
            n=n+1;
            ysum=ysum+matxParaMatx(rowParaMatx,colParaMatx);
            y2sum=y2sum+matxParaMatx(rowParaMatx,colParaMatx)^2;
        end
    end
    matxSampleMS(rowParaMatx,1)=n;
    matxSampleMS(rowParaMatx,2)=ysum;
    matxSampleMS(rowParaMatx,3)=y2sum;
end
matxOutSampleMS=matxSampleMS;

%Main function
function [matxOutQTL, matxOutBestLog, matxOutActPro]=. . .
funcQTL2(intArgTimeDiffStart, intArgTimeStoc, matxArgMarkerFull,. . .
matxArgResp, intNoTimesGS, intArgBP, doubleArgBetaVar, . . .
doubleArgThetaVarNull, doubleArgSigmaNull)
    matxSizeMarkerFull=size(matxArgMarkerFull);
    vecSizeResp=size(matxArgResp);
    intNoMarker=matxSizeMarkerFull(1,2);
```

```
    intNoLine=vecSizeResp(1,1);

    matxOutQTL=sparse(2^intNoMarker,4);

    matxBestMarker=[];
    intBestModelValue=-999;

    matxOutBestLog=zeros(intArgTimeStoc,intArgTimeDiffStart);

    %doubleNowAvgLLike=0;
    doubleBestAvgLLike=0; %Not necessary

    [vecTheta,matxRespMS]=funcThetaInit(matxArgResp);
    doubleThetaVar=funcThetaVarInit(vecTheta);
    vecSigma=funcSigmaInit(matxRespMS,intNoLine);

    vecRgrnResp=funcRgrnResp(matxArgResp,matxRespMS,intNoLine);

    doubleThetaVarAlpha=doubleArgThetaVarNull+intNoLine/2;
    vecSigmaAlpha=ones(intNoLine,1)*doubleArgSigmaNull . . .
+(matxRespMS(:,1)/2);
    matxModelFull=funcModelFull(intNoMarker,intNoLine);
    doubleAdj=funcLLike(matxModelFull, matxArgMarkerFull, . . .
vecRgrnResp, matxRespMS, intNoLine, intNoMarker, . . .
intNoTimesGS, intArgBP, doubleThetaVar, vecSigma, . . .
doubleThetaVarAlpha, vecSigmaAlpha, doubleArgBetaVar, 0, 0)
```

```
    for intLoopDS=1:1:intArgTimeDiffStart
        [intModelVal, matxModelSlt]=funcModelSelect(intNoMarker);

        while(matxOutQTL(intModelVal+1,2)==1)
            [intModelVal, matxModelSlt]=funcModelSelect(intNoMarker);
        end
        matxOutQTL(intModelVal+1,2)=1;


       for intLoopStoc=1:1:intArgTimeStoc
            disp 'funcQTLV2.m(117)new stochastic';
            disp '======================';
            intLoopDS
            intLoopStoc
            intBestModelValue
            intModelVal
            disp '======================';
            if(matxOutQTL(intModelVal+1,3)~=0)
                doubleNowAvgLLike=matxOutQTL(intModelVal+1,1);
            else
                doubleNowAvgLLike=funcLLike(matxModelSlt, . . .
matxArgMarkerFull, vecRgrnResp, matxRespMS, intNoLine, intNoMarker, . . .
intNoTimesGS, intArgBP, doubleThetaVar, vecSigma, . . .
doubleThetaVarAlpha, vecSigmaAlpha, doubleArgBetaVar, doubleAdj, 1);
                matxOutQTL(intModelVal+1,1)=doubleNowAvgLLike;
                matxOutQTL(intModelVal+1,3)=1;
            end
```

```
            if intLoopStoc==1
                doubleBestAvgLLike=doubleNowAvgLLike;
                matxBestMarker=matxModelSlt;
                intBestModelValue=intModelVal;
                matxOutBestLog(1,intLoopDS)=intBestModelValue;
                matxOutQTL(intModelVal+1,4)=matxOutQTL(intModelVal+1,4)+1;
                [intModelVal, matxModelSlt]=. . .
funcModelSwitch(intBestModelValue, matxBestMarker);
            else
                if(doubleBestAvgLLike~=0)
                    doubleProMove=doubleNowAvgLLike/doubleBestAvgLLike;
                else
                    if doubleNowAvgLLike>0
                        doubleProMove=doubleNowAvgLLike;
                    else
                        doubleProMove=0;
                    end
                end



                if(doubleProMove>1)
                    doubleProMove=1;
                end

                if(rand(1)<doubleProMove)
                    intDec=1;
```

```
                else
                    intDec=0;
                end

                if(intDec==1)
                    disp 'current model is better than the best one';
                    matxBestMarker=matxModelSlt;
                    intBestModelValue=intModelVal;
                    matxOutBestLog(intLoopStoc,intLoopDS)=. . .
intBestModelValue;
                    matxOutQTL(intModelVal+1,4)=. . .
matxOutQTL(intModelVal+1,4)+1;
                    [intModelVal, matxModelSlt]=. . .
funcModelSwitch(intBestModelValue, matxBestMarker);
                else
                    disp 'current model is worse than the best one';
                    matxOutBestLog(intLoopStoc,intLoopDS)=. . .
intBestModelValue;
                    matxOutQTL(intBestModelValue+1,4)=. . .
matxOutQTL(intBestModelValue+1,4)+1;
                    intNoModelValuePre=intModelVal;
                    while(intNoModelValuePre==intModelVal)
                        [intModelVal, matxModelSlt]=. . .
funcModelSwitch(intBestModelValue, matxBestMarker);
                    end
                end
            end
```

```
        end
    end

    matxOutActPro=sparse(1,intNoMarker);
    for intValModel=0:1:(2^intNoMarker-1)
        if(matxOutQTL(intValModel+1,3)~=0)
            intCurValModel=intValModel;
            for j=0:1:(intNoMarker-1)
                weight=2^(intNoMarker-j-1);
                if(intCurValModel<weight)
                   continue;
                else
                     matxOutActPro(1,j+1)=. . .
matxOutActPro(1,j+1)+matxOutQTL(intValModel+1,4);
                     intCurValModel=intCurValModel-weight;
                end
            end
        end
    end
    matxOutActPro=matxOutActPro/(intArgTimeDiffStart*intArgTimeStoc);
save([datestr(now,'yyyymmddHHMMSS'),'.mat'],. . .
'matxOutQTL', 'matxOutBestLog', 'matxOutActPro','doubleAdj','-mat');
```